VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky

BAKALÁŘSKÁ PRÁCE

2017 **Jan Hájek**

-

VŠB – Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Možnosti implementace řídících algoritmů na mikrokontroléru Arduino s použitím prostředí IDE, MatlabSimulink a SciLab

Possibilities of Implementation of Control Algorithms on MCU Arduino with the Use of IDE, MatlabSimulink and SciLab

2017 Jan Hájek

VŠB - Technická univerzita Ostrava
Fakulta elektrotechniky a informatiky
Katedra kybernetiky a biomedicínského inženýrství

Zadání bakalářské práce

Student: **Jan Hájek**

Studijní program: B2649 Elektrotechnika

Studijní obor: 2612R041 Řídicí a informační systémy

Téma: Možnosti implementace řídicích algoritmů na mikrokontroléru Arduino s použitím prostředí IDE, Matlab&Simulink a SciLab
Possibilities of Implementation of Control Algorithms on MCU Arduino with the Use of IDE, Matlab&Simulink and SciLab

Jazyk vypracování: čeština

Zásady pro vypracování:

1. Seznámení se s mikrokontrolérem Arduino Uno.
2. Seznámení se s prostředími IDE a s nástroji v prostředích Matlab&Simulink a SciLab určenými pro ovládání mikrokontroléru Arudino.
3. Rešerše vlastností jednotlivých způsobů realizace řídicích algoritmů se zaměřením na jejich užitné, archivační a vizualizační možnosti.
4. Případová studie – realizace řízení zvolené regulované soustavy výše uvedenými způsoby, porovnání řešení.
5. Zhodnocení dosažených výsledků závěrečné práce, závěr.

Seznam doporučené odborné literatury:

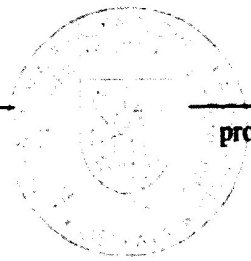
[1] XUE, Dingyü, Yangquan CHEN a Derek P. ATHERTON. *Linear feedback control: analysis and design with MATLAB*. 4th rev. and enlarged ed. Philadelphia, PA: Society for Industrial and Applied Mathematics, c2007. xii, 354 p. ISBN 08-987-1638-1/978-0-898716-38-2.
[2] NOSKIEVIČ, Petr. *Modelování a identifikace systémů*. 1. vyd. Ostrava: MONTANEX, a. s., 1999. 276 s. ISBN 80-7225-030-2.
[3] VÍTEČKOVÁ, Miluše a Antonín VÍTEČEK. *Základy automatické regulace*. 2. vyd. Ostrava: VŠB-TU Ostrava, 2008. 243 s. ISBN 978-80-248-1924-2.
[4] VAVŘÍN, Petr. *Teorie automatického řízení I (Lineární spojité a diskrétní systémy)*. 2. přepracované vyd. Brno: VUT Brno, 1991. 158 s. ISBN 80-214-0244-X.
[5] VAVŘÍN, Petr. *Teorie dynamických systémů*. 1. vyd. Brno: VUT Brno, 1989. 177 s.
[6] ZÍTEK, P., M. HOFREITER a J. HLAVA. *Automatické řízení*. Vyd. 2., přeprac. Praha: Vydavatelství ČVUT, 1999. 148 s. ISBN 80-010-2044-4.
[7] ŠULC, Bohumil. *Teorie automatického řízení s počítačovou podporou*. Vyd. 1. Praha: ČVUT, Strojní fakulta, 1999. 154 s. ISBN 80-010-1974-8.
[8] OŽANA, Štěpán. *Navrhování a realizace regulátorů*. vyd. 1. Ostrava: VŠB - TU Ostrava, 2012. Studijní materiály. 136 s. ISBN 978-80-248-2605-9.
[9] ZEZULKA, František, Petr FIEDLER a Zdeněk BRADÁČ. *Prostředky průmyslové automatizace*. Učební texty. Brno: VUT v Brně, 2002.
[10] Firemní dokumentace Matlab and Simulink (MathWorks).
[11] Firemní dokumentace Rex Controls.
[12] Firemní dokumentace Arduino Uno.

Formální náležitosti a rozsah bakalářské práce stanoví pokyny pro vypracování zveřejněné na webových stránkách fakulty.

Vedoucí bakalářské práce: **doc. Ing. Štěpán Ožana, Ph.D.**

Datum zadání: 01.09.2016

Datum odevzdání: 28.04.2017



doc. Ing. Jiří Koziorek, Ph.D.
vedoucí katedry

prof. RNDr. Václav Snášel, CSc.
děkan fakulty

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně. Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

V Ostravě 2017 ..... Hájek .....

Rád bych na tomto místě poděkoval všem, kteří mi s prací pomohli, protože bez nich by tato práce nevznikla.

Abstrakt

Cílem této bakalářské práce je seznámení s možnostmi řízení mikrokontroléru Arduina Una v prostředí Matlab, Scilab a IDE.

V úvodu práce je popis mikrokontroléru Arduina Una a jeho specifikace. Následující kapitoly jsou věnovány realizaci základních řídících operací s miktrokontrolérem Arduino Uno v jednotlivých prostředích (čtení a zápis digitální a analogové hodnoty, 1-Wire komunikace, PWM a další).

V práci je také realizován příklad identifikace přenosu a regulace stejnosměrného motoru.

Závěrem práce je porovnání možností implementace algoritmů v jednotlivých prostředí.

Klíčová slova: Arduino Uno, Matlab, Scilab, Arduino IDE, Sketch, Simulink

Abstract

The aim of this bachelor thesis is to introduce the possibilities of Arduin Una microcontroller control in Matlab, Scilab and IDE.

At the beginning of the thesis is the description of Arduina Una microcontroller and its specification. The following chapters deal with basic Arduino Uno microprocessor control operations in individual environments (reading and writing of digital and analog values, 1-Wire communication, PWM and others).

An example of transfer and control identification of a DC motor is also realized.

The conclusion of the thesis is the comparison of possibilities of algorithm implementation in individual environments.

Key Words: Arduino Uno, Matlab, Scilab, Arduino IDE, Sketch, Simulink

Obsah

Seznam použitých zkratek a symbolů

IDE	–	Integrované vývojové prostředí
CTC	–	Clear Timer on Compare Match
GPIO	–	General purpose input/output
LSB	–	Least significant bit(nejméně významný bit)
GUI	–	Graphical User Interfaces(Grafické uživatelské rozhraní)

Seznam obrázků

Seznam tabulek

Seznam výpisů zdrojového kódu

1 Úvod

Arduino je otevřená vývojová patforma, založená na uživatelsky nenáročném hardwaru a softwaru. Prvotním účelem Arduina bylo vytvoření platformy, jenž by byla vhodná pro studenty jak po cenové, tak i po uživatelské stránce. Tato platforma se v průběhu nekolika málo let stala velmi populární, zejména mezi studenty a rozrostla se o množství desek, shieldů a vstupní a výstupní periférie. Tato práce se zabývá možnostmi implementace kódu konkrétně na mikrokontroléru Arduino Uno v prostředích Matalab, Scilab a Arduino IDE. Prostředí Matlab a Scilab jsou primárně určeny pro matematické výpočty, modelování a simulaci, nicméně nedávno byla pro tyto prostředí vyvinuta podpora platformy Arduino v podobě tzv. hardwarových balíčků. Prostředí Arduino IDE je prostředí vyvinuté společností Arduino software a na rozdíl od dvou zmíněných je určené přímo pro programování desek Arduina.

V práci jsou popisány jednotlivá prostředí, zejména nástroje týkající se programování Arduina. Dále jsou v práci uvedeny příklady použití některých nástrojů, a je zde realizován příklad identifikace a řízení soustavy stejnosměrného motoru v jednotlivých prostředích.

Cílem této bakalářské práce je tedy seznámení s platformou Arduino Uno a poukázaní na možnosti implemetace kódu v zmíněných prostředích.

2 Arduino

Adruino je otevřená nízko-nákladová platforma, vyznačující se uživatelsky jednoduchým hardwarem a softwarem určená pro studenty, vývojáře a kutily. Vznik projektu Arduino předcházela diplomová práce Hernanda Barragána, jejímž cílem bylo usnadnění práce studentům a designérům pracujících s elektronikou. Výstupem této práce bylo tzv. zařízení Wiring. Tato platforma se později začala prodávat po celém světě. V roce 2005 se od projektu Wiring odpojili Massiomo Banzi a David Mellis, kteří společně s Davidem Cuartiellesem, Tomem Igoem a Gianlucem realizovali projekt Arduino. Ačkoliv arduino samo o sobě nic nového na trh nepřineslo, stalo se během posledních let hotovým fenoménem (mezi roky 2005-2013 se prodalo přes 700 000 oficiálních zařízení Arduino). Příčiny tohoto úspěchu jsou [1] :

- jednoduché a srozumitelné vývojové prostředí, které se stará o všechno potřebné bez nutnosti zásahu uživatele (vhodné pro studenty, disignéry a lidi z netechnických oborů, kteří mají omezené znalosti s programováním hardwaru).
- Široká podpora komunity (ta se zpočátku formovala hlavně díky silnému Marketingu)
- Relativně nízká cena
- Open-source projekt, zahrnující jak software tak i hardware (volně dostupné IDE, schémata, diagramy).
- USB připojení. Nevyžaduje programátor a znalost jeho používání.
- Rychlý vývoj, a dostupnost modulů (Shieldů) v podobě snímačů, konvertorů mezi komunikačními protokoly, atd., které rozšiřují možnosti použití Arduina.
- Existence několika virtuálních emulátorů a simulátorů, nahrazující část funkcionality reálné desky Arduina(např. CodeBlocks Arduino IDE, Simuino,VBB4Arduino).

Nicméně Arduino má i řadu nevýhod jako:

- Produkty platformy Arduino nedisponují příliš dlouhou životností
- Arduino není vhodné pro komerční aplikace, jelikož ty většinou vyžadují certifikaci ke splnění požadávků
- Platforma v součastnosti nenabízí možnost použití šifrovacích protokolů.
- V posledních letech se na trh dostávají nekvalitní produkty, vyráběné převážně v Číně.

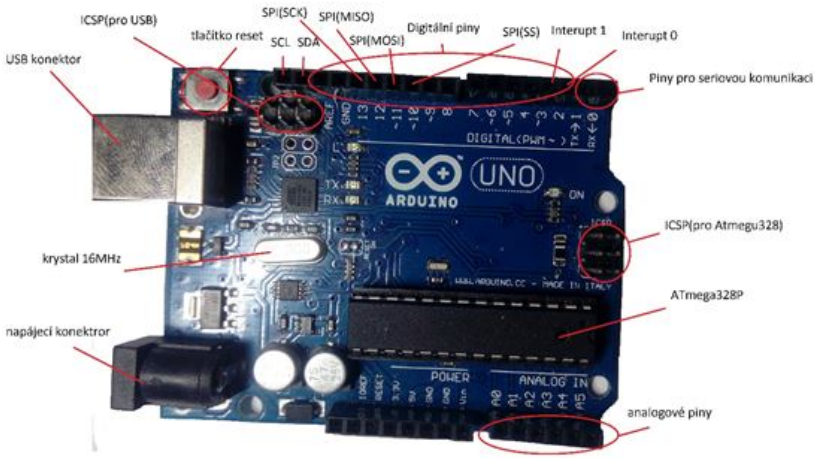
Velkým přínosem pro studenty je programovací prostředí Arduino IDE, které používá jazyk C doplněný o knihovny, určené pro desky Arduina. Pomocí těchto knihoven se uživatel vyhne přímému programování registrů, které vyžaduje větší znalosti a je méně přehledné. Další usnadněním je již nahraný bootloader.[1]

2.1 Arduino Uno

Arduino Uno je deska vyráběná Italskou firmou Smart Projects používající čip ATMega 328. Tato deska byla uvedena na trh v roce 2010 a od té doby se stala nejprodávanější deskou Arduina. Deska má už předprogramovaný bootloader(zavaděč), ten umožňuje nahrání nového kódu bez použití externího hardwarového programátoru. Komunikace probíhá pomocí protokolu STK500 přes UART0 rychlostí 115200 Bd.[9]

Flash paměť	32kB
SRAM	2kB
EEPROM	1kB
General Purpose I/O	23
Architektura	Atmel AVR
Taktovací frekvence	16MHz
SPI	2
I2C	1
USART	1
ADC	10-bitové
ADC kanály	8
Časovače(8-bitové)	2
Časovače(16-bitové)	1
Hodiny	16Mhz
Operační napětí	5V
Digitální I/O piny	14
PWM piny	6
Analogové piny	6

Tabulka 1: Specifikace mikrokontroléru

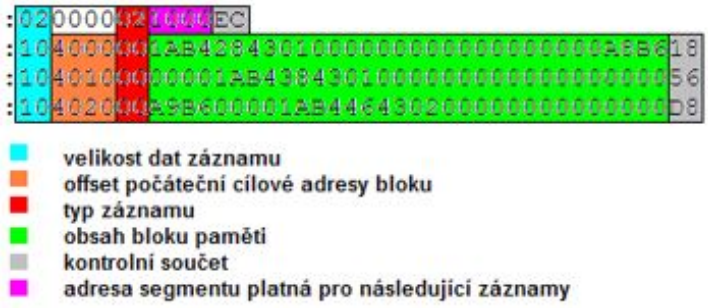


Obrázek 1: Mikrokontrolér Arduino Uno

3 Arduino IDE

Nejrozšířenější integrované programovací prostředí pro vytváření programů pro platformy Arduina. Jedná se open-source prostředí, tedy prostředí s otevřeným kódem, volně dostupným z GitHubu. Arduino IDE běží na Windows, Mac Os X a Linuxu. Programovací jazyk je Wiring. Jedná se v podstatě o C/C++ rozšířený o knihovny, usnadňující programování Arduina(použití knihoven se uživatel vyhne přímému zapisování do registrů). Arduino IDE je napsáno v jazyku Java a vychází z Proccesingu a z jiných open-source. Uživatel v tomto prostředí vytváří tzv. Sketche, soubory s koncovku *soubor*.**ino** u starších verzích *soubor*.**ide** (dnes se již nepoužívá).

Proces kompilace je v tomto prostředí realizovám pomocí $avr-gcc$, kompilátoru od fimy Intel. Během kompilace dojde nejprve k překladu a optimalizaci kódu, vstupem jsou soubory s koncovnkou *soubor*.**c**, *soubor*.**cpp**, *soubor*.**h** a *soubor*.**ino**(což je hlavní soubor vygenerovaná prostředím Arduino IDE). Výstupem je soubor s koncovkou *soubor*.**o**. Proces kompilace(výpisy o průběhu a chybové výpisy) lze sledovat z konzole, je-li výpis kompliace povolen(Soubor->Vlastnosti). V následujícím kroku se vytvoří *soubor*.**elf**, který se poté převede do souboru *soubor*.**hex**, což je již spustitelný soubor. Soubory vygenerované během procesu kompilace lze obvykve najít ve složce Temp. Soubor *soubor*.**hex** je formát souboru, v nemž jsou data kódována v šestnástkové soustavě viz obrázek 2[7].[1]



Obrázek 2: Popis hex souboru

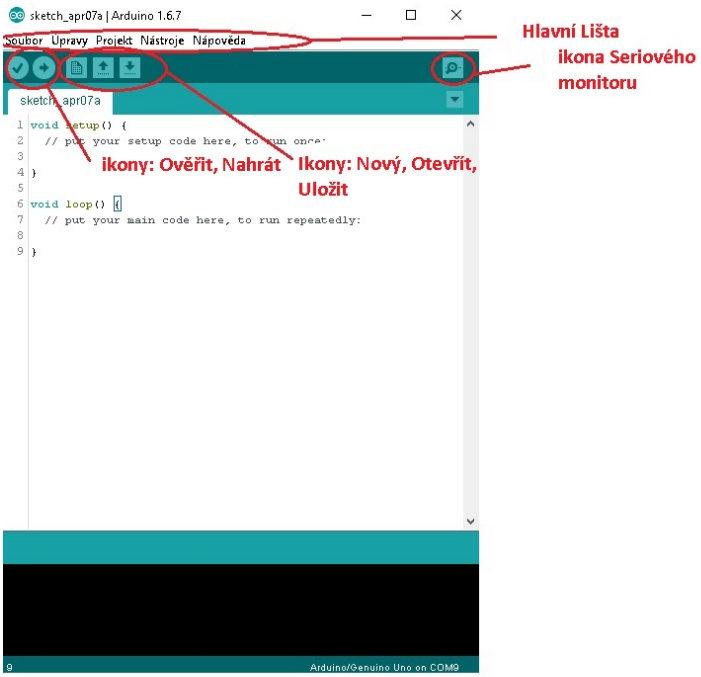
Data jsou formována do záznamů, které jsou psány vždy po jednom na jeden řádek a mají pevně danou strukturu: řádek vždy začína dvojtečkou, první dvojice znaků definuje počet bajtů, další čtyři definují adresu, následující dva znaky definují typ dat(00-samotná data, 01-záznam neobsahuje data a končí), zbylé znaky kódují data a poslední dva znaky udávají kontrolní součet. Soubor HEX je poté prostředím Arduino IDE nahrán pomocí nástroje AVRdude přes sériovou linku do Arduina. Proces nahrávání končí uložením řídícího kódu do flash paměti. U nových mikrokontrolérů se ještě testuje reset příznak ve flash paměti. Pokud příznak není nastaven zkopíruje se obsah flash paměti do paměti RAM(kvůli rychlejšímu přístupu). Poté se do paměti RAM kopírují data ze sekce .data. Následuje kopírování sekce .bss. Po dokončení kopírování mikrokontrolér skočí na začátek paměti a začne vykonávat program.[1]

Prostředí Arduino IDE obsahuje také nástroje pro vypálení bootloaredu, což je program, který kontroluje zda data nepřesahují velikost paměti a pokud ne umistuje program do programové paměti. Nicméně bootloader je již u většiny desek Arduina zaveden a tím odpadá povinost jej zavádět. Pro jeho vypálení je totiž nutný tzv. programátor. Dalšími nástroji jsou Serioý monitor a ploter, jenž jsou popsány níže.

3.1 Popis prostředí Arduino IDE

Prostředí se skládá z textového okna pro vytváření tzv. Sketche, hlavní lišty a lišty obsahující ikony, sloužící k ověření souboru, nahrání kódu, otevření a uložení Sketche a seriový monitor viz. obrázek 3. V prostředí jsou dva nástroje sloužící k zobrazení hodnot přes sériovou linku. A to Seriový monitor, ten slouží k výpisu dat ze sériové linky a Seriový Polter, který zobrazuje data ve formě grafu.

Prostředí Arduino IDE velmi usnadňuje práci s knihovnami, v záložce **Projekt->Přidat knihovnu** je seznam přidaných knihoven a jsou zde položky **Spravovat knihovny**, pomocí níž lze přidat nové knihovny dostupné na internetu, a **Přidat .zip knihovnu**,která přidá zip balíček s knihovnou. V záložce **Soubor** je položka **Příklady**, která zobrazí dostupné příklady použití jednotlivých knihoven. Před nahráním hotového programu je nejprve nutné v Arduinu IDE nastavit typ desky a port na kterém je připojena, to se provádí v záložce **Nástroje->Vývojová deska** a **Port**. V dolní části okna Arduina IDE je umístěna konzole, sloužící pro výpis chyb a průběhu kompilace.



Obrázek 3: Arduino IDE

3.1.1 Programování v prostředí Arduino IDE

Jak již bylo zmíněno programovacím jazykem prostředí Arduino IDE je jazyk C/C++ rozšířený o knihovny určené pro mikrokontroléry Arduino. Základní knihovny v Arduinu IDE definují funkce pro sériovou komunikaci, manipulaci s digitálními a analogovými piny, časování, ISP, PWM, práci se textovým řetězcem atd.

3.1.1.1 Základní funkce v prostředí Arduino IDE

- **void setup():** metoda, která se vykoná jednou po spuštění programu. Ve funkci je vhodné provést nastavení pinů a prvotní inicializaci počátečních hodnot proměných)
- **void loop():** metoda, provádějící se opakovaně po spuštění programu.

3.1.2 Definování vstupů a výstupů

K definování pinů v prostředí Arduinu IDE lze použít funkci **pinMode(pin,smer)**. Tato funkce má dva povinné parametry: pin, značící číslo pinu(digitální piny jsou určeny číslem pinu a analogové jsou definovány A0-A5, pro Arduino Uno) a parametr smer, který může nabývat hodnot INPUT(nastaví daný pin jako vstupní), OUTPUT(nastaví daný pin jako výstupní) a INPUT_PULLUP (nastaví daný pin jako vstupní a připojí pull-up rezistor).

```
void setup()
{
pinMode(2,OUTPUT);
pinMode(A1,INPUT);
pinMode(3,INPUT_PULLUP);
}
```

Výpis 1: Nastavení pinů

Piny lze definovat také přímým zápisem do registrů, defaultně jsou piny nastaveny jako INPUT . Atmega 328 má tři osmibitové porty B (digitální piny 8-13), C(analogové piny) a D(digitální piny 0-7).Každý port je ovládán pomocí tří registrů:

- DDRx : Data direction Registr
- PORTx : Pin Output Registr
- PINx : PinInput Registr

Znak x definuje o jaký port se jedná (za x se identifikátor příslušného portu, tedy B, C nebo D). Registr DDRx=Bnnnnnnnn (n určuje příslušný pin) určuje směr toku. Zápisem logické 1 na libovolnou pozici x definujeme daný pin jako OUTPUT, zápisem 0 na x definujeme pin jako INPUT). PORTx=Bnnnnnnnn(n určuje stav pinu) definuje hodnoty výstupních pinů.Zápisem 1 definujeme stav HIGH(logická 1) a zápisem 0 stav LOW(logicka 0)). PINx slouží pro čtení

hodnot pinů na daném portu. Zapsáním logické 1 na bit PUD v registru MCUCR lze odpojit pull up rezistory.[4]

```
 void setup() {
 DDRB=B00000001; //8 pin jako Output
 PORTB=B00000000;
}

void loop() {
PORTB=B00000001;
delay(500);
PORTB=B00000000;
delay(500);
}
```

Výpis 2: Příklad ovládání LED diody pomocí přímého zápisu do registrů

```
bool stav;
void setup() {
  DDRD=B00001000; //pin 3 jako Input
  Serial.begin(9600);
  while(!Serial){
    ;
  }
}

void loop() {
stav = PIND & (1 << 3);
Serial.println(stav);
delay(100);
}
```

Výpis 3: Čteni stavu vstupního pinu pomocí přímého zápisu do registrů

Manipulace s piny pomocí zápisu do registrů je v porovnání s funkcemi jako je pinMode(), digitalWrite() atd. mnohem rychlejší a méně náročná na paměť nicméně program se stává méně čitelným a je větší pravděpodobnost výskytu chyby (při nesprávné konfiguraci pinu 0 a 1 může dojít k selhání sériové komunikace,tuto piny totiž slouží k sériové komunikaci). Dále je nutné dávat pozor, aby součet proudů pro spínání zátěže pro všechny proudy nepřesáhl 200mA.

3.1.3 Čtení a zapisování hodnot

- **Digitální piny:** Pro čtení a zápis slouží funkce digitalRead(pin) a digitalWrite(pin,stav). Obě funkce vyžadují parametr pin (číslo pinu, nebo proměnná, která uchovává číslo pinu). Funkce digitalWrite() vyžaduje určení stavu, který se zapíše na výstup (tedy logická 1=HIGH, nebo logická 0=LOW). Pro čtení a zápis můžeme využít přímý přístup pomocí registrů PORTn(zápis) a PINn(čtení) popsaných výše.

- **Analogové piny:** Arduino Uno disponuje šesti analogovými piny, které mohou sloužit také jako GPIO piny. Piny jsou napojeny na A/D převodník s rozlišením 10bitů. Pro manipulaci s nimi slouží funkce analogRead(pin) a analogWrite(pin,hodnota). Parametr pin, se pro analogové piny, zapisuje jako Ax, kde x je číslo daného pinu. Parametr hodnota může nabývat hodnot v rozmezí 0 až 255 (0 – 5V). Výstupem funkce analogRead je hodnota v rozmezí 0 až 1023. Funkce analogWrite() využívá PWM ke generování požadovaného napětí na výstupu, proto lze použít i u digitálních pinů určených pro PWM(piny 3,5,6,9,10,11).

3.2 Sériová komunikace

K sériové komunikaci se u Adrunia používají piny TX a RX (piny používají TTL logickou úroveň a jsou tedy dimenzovány na 5V nebo 3.3V záleží to na typu desky, proto je nemůžeme rovnou připojit na RS232 port, který je napájen 12V). Pro sériovou komunikaci má Arduino Unu piny Rx a Tx (piny 0 a 1), proto se při používání sériové komunikace nesmí použít piny 0 a 1 pro digitální vstup nebo výstup. Sériová komunikace se dá použít ke komunikaci s jiným zařízením jako je raspberry pi, senzory používající sériovou komunikaci či jiné arduino. Sériovou komunikaci lze použít i pro zobrazování hodnot ve Sketchi pře Serial Monitor nebo vykreslování grafu hodnot přes Serial Plotter. Sériová komunikace je v Arduinu IDE nejsnadněji realizovatelná pomocí funkcí **Serial.begin(rychlost)**,**Serial.Write(String)** a **Serial.Read()**.

```
byte analogPin = A0;
int hodnota;
void setup() {
pinMode(analog,Input);
Serial.begin(9600);
while(!Serial) {
  ;
}
}

void loop() {
hodnota=analogRead(analogPin);
Serial.println(zprava);
delay(500);
}
```

Výpis 4: Čtení analogové hodnoty se zobrazením přes serial plotter

Obrázek 4: Zobrazení průběhu v Serial plotteru

3.3 Časovače

Arduino disponuje třemi časovači: Timer0, Timer1 a Timer2, z čehož Timer0 a Timer2 je 8-bitový a Timer1 je 16-bitový. To znamená , že Timer0 a Timer2 může čítat jen do hodnoty 255, kdežto Timer2 je může čítat až do hodnoty 65 535. Časovač funguje v principu tak, že do registru TCNTn (kde n značí číslo časovače, např. n=1 pro Timer1) čítá pulsy z hodin Arduina(16MHz) nebo externích hodin a buď porovnává tento počet s registrem OCRnx(x značí výstupní komparační jednotku A/B), ve kterém se nastavuje hodnota, při které dojde k vystavení příznaku přerušení nebo dojde k přetečení registru TCNTn a vystavení příznaku přerušení, jakmile dojde k přetečení(je nutné si uvědomit, že čítač může čítat v obou směrech). Příznak přerušení se automaticky smaže po vykonání příslušné obslužné rutině přerušení. Frekvence čítání lze upravit nastavením tzv. prescalarní hodnoty, jíž se pak dělí frekvence hodin.Tato hodnota se nastavuje v registru TCCRxn. Doba mezi jednotlivými ikrementacemi registru TCNTn je dána rovnicí 1, kde p značí prescalární hodnotu a f frekvenci hodin.[4]

$$T = p\frac{1}{f} \tag{1}$$

CA02	CA01	CA00	Popis
0	0	0	Timer zastaven
0	0	1	Prescalarní hodnota = 1
0	1	0	Prescalarní hodnota = 8
0	1	1	Prescalarní hodnota = 64
1	0	0	Prescalarní hodnota = 256
1	0	1	Prescalarní hodnota = 1024
1	1	0	Externí hodiny(pin T0).Sestupná hrana.
1	1	1	Externí hodiny(pin T0).Vzestupná hrana.

Tabulka 2: Nastavení prescalarni hodnoty

Časovače je možné nastavit do jednoho ze čtyř základních módů časovače(Normal,Fast PWM, CTC, Phase Correct PWM), toto nastavení se provádí v registrech TCCRnA a TCCRnB na bitech WGMn0 až WGMn2. V PWM módech se časovač používá ke generování PWM signálu. Nicméně pokud nechceme časovače nastavovat přímým zápisem do registrů, můžeme využít některou z dostupných knihoven pro časovače např. TimerOne, MsTimer2 nebo TimerThree. Příklad demonstrující použití knihoven TimerOne a MsTimer2.

```
#include<TimerOne.h>
#include<MsTimer2.h>
#define PERIODA_1 100000 //10ms -> Timer1 se nastavuje s mikrosekundach
#define PERIODA_2 5000 //5s -> Timer2 se nastavuje v milisekundach
byte ledka = 11;
void setup() {

  // inicializace casovacu
pinMode(ledka,OUTPUT);
Timer1.initialize(PERIODA_1);
Timer1.attachInterrupt(OR_Timer1);
MsTimer2::set(PERIODA_2,OR_Timer2);
MsTimer2::start();
}

void loop() {

}

void OR_Timer1() {
  digitalWrite(ledka,digitalRead(ledka)^1);//zmeni se stav ledky
}
void OR_Timer2() {
  //v obsluzne rutine se vypne Timer1
  Timer1.detachInterrupt();
  MsTimer2::stop();
}
```

Výpis 5: Příklad použití knihoven TimerOne a MsTimer2 pro ovládání led doidy

```
volatile bool stav = false;
//program pro demonstraci pouziti Timeru 1 v modu Normall pro blikani ledky
//v intervalu 2s
void setup()
{
 DDRB=B00000001; //8 pin jako Output
 PORTB=B00000000;

  // inicializace casovacu
  SREG=0; //zablokovani preruseni
  TCCR1A = 0;
```

```
  TCCR1B = 0;

  TCNT1 = 3035;          // hodnota,do ktere se cita
  TCCR1B |= (1 << CS12); // precsalarni hodnota
  TIMSK1 |= (1 << TOIE1); // povoleni preruseni pri preteceni
  SREG=B10000000;                //povoleni preruseni
}

ISR(TIMER1_OVF_vect)     // obsluzna rutina preruseni
{
  TCNT1 = 3035;          // opetovne nastaveni hodnoty do ktere se cita
  PORTB = (stav << 0);//zmena stavu pinu s led diodou
  stav = !stav;
}

void loop()
{

}
```

Výpis 6: Příklad použití Timeru1 v režimu Normal (pomocí zápisu do registrů)

4 Externí přerušení

Externí přerušení je druh přerušení, u kterého se příznak přerušení vystaví pokaždé, když dojde k přednastavené změně napětí(falling edge, rising edge) na pinu, který umožňuje externí přerušení(u Arduina Una je to pin 2 a 3). Externí přerušení se používá v případě, kdy je nutné reagovat na vnější asynchronní události, nicméně přerušení lze vyvolat i na pinu definovaném jako výstupní, což umožňuje vyvolávat softwarové přerušení. Externí přerušení se v Arduinu IDE nastavují pomocí funkce **attachInterrupt (digitalPinToInterrupt(pin), rutinaPřerušení())**, tato funkce má parametr specifikující pin, který má inicializovat přerušení, tedy pro Arduino Uno zde zapisujeme 0 nebo 1, kde 0 = pin2 a 1= pin3, nebo můžeme použít funkci **digitalPinToInterrupt()**, do které již přímo zadáváme pin2 nebo pin3. Další parametr funkce je obslužná rutina, zde se jedná o metodu, která se provede, jakmile se vystaví příznak přerušení. Užitečnými funkcemi jsou funkce **sei()** nebo **interrupts()**,povolující globálně přerušení a funkce cli() nebo **detachInterrupts()**, zakazující přerušení. Tyto funkce je vhodné používat k ošetření kritických sekcí např. pro ošetření probíhajícího přenosu dat přes sériovou komunikaci v programu, kde muže dojít k některému z přerušení.

```
//Priklad pouziti externiho preruseni, jakmile dojde k detekci sestupne hrany
//na pinu 2 zmeni se stav ledky. Dojde-li k detekci vzestupne hrany na pinu 3
//vypise se do serioveho monitoru "priklad kriticke sekce"
byte pin2=2;
byte pin3=3;
byte ledka = 11;
void setup() {
  Serial.begin(9600);
  pinMode(ledka,OUTPUT);
  pinMode(pin2,INPUT);
  pinMode(pin3,INPUT);
  attachInterrupt(digitalPinToInterrupt(pin2),OR_pin2,FALLING);
  attachInterrupt(1,OR_pin3,RISING);

}

void loop() {
}
//obsluzne rutiny preruseni OR_pin3 a OR_pin2 se vykonaji jakmile je
//vystaven prislusny priznak preruseni
void OR_pin3() {
  digitalWrite(ledka,digitalRead(ledka)^1);
}
void OR_pin2() {
  cli();//globalni zablokovani preruseni
  Serial.println("priklad kriticke sekce");
  sei();//globalni povoleni preruseni
}
```

Výpis 7: Příklad implementace kódu s externím přerušením(pin je definován jako vstupní)

```
#define INTERRUPT_PIN 2
#define LEDKA 8
unsigned long Start = 0;
bool stav = 0;
void setup() {
  pinMode(LEDKA, OUTPUT);
  pinMode(INTERRUPT_PIN, OUTPUT);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), blikej, RISING);
  Start = millis();
}

void loop() {
  if (millis() - Start >= 1000) {
    digitalWrite(INTERRUPT_PIN, (digitalRead(INTERRUPT_PIN)) ^ 1);
    Start = millis();
  }
}
void blikej() {
  digitalWrite(LEDKA, digitalRead(LEDKA) ^ 1);
}
```

Výpis 8: Příklad softwarového přerušení pomocí pinu definovaného jako výstupní

Pro přímý zápis do registrů v případě externího přerušení je nutné použít registry SREG, EICRA, EIMSK a EIFR.[4]

- **SREG:** Tímto registrem globálně povolujeme přerušení tak, že se na sedmém bitu registru zapíše 1, v případě 0 se přerušení globálně zakazuje.[4]
- **EICRA:** Registrem se nastavuje způsob, jakým se vystaví příznak přerušení(tedy jestli se přerušení vystaví na vzestupnou hranu, sestupnou hranu nebo na změnu stavu). Bity ISC11 a ISC01 definují externí přerušení 1 a bity ISC01 a ISC00 definují externí přerušení 0. V tabulce je znázorněno definování režimů spuštění externích přerušení.[4]
- **EIMSK:** Zapsáním logické 1 na pozici 0 a 1(bity INT0 a INT1) se povoluje dané externí přerušení.[4]
- **EIFR:** Do tohoto registru se na bit INTF1 a INTF0 zapíše logická jednička, jakmile dojde k splnění podmínky pro přerušení.[4]

ISCn1	**ISCn0**	**význam**
0	0	logicka 0 generuje požadavek přerušení
0	1	jakakoliv zmena logické hodnoty genruje přerušení
1	0	generuje přerušení při sestupné hraně
1	1	generuje přerušení při vzestupné hraně

Tabulka 3: Nastavení řežimu externích přerušení

```
//ovladani ledky pomoci externiho preruseni, reagujici na vzestupnou
//hranu na pinu 2, pin 2 je definovan jako vystup a pomoci casovace je
//menena jeho hodnota(vyvolava se tedy softwarove preruseni)
volatile bool stav = false;
volatile bool pin2 = false;
void setup() {
  DDRD = B00000100;
  EIMSK = B00000001; //preruseni nastaveno na pin 2
  EICRA = B00000011; //nastaveni preruseni na vzestupnou hranu
  DDRB = B00000001; //nastaveni pinu 8 jako vystupu(pro ovladani led)

  // inicializace casovacu
  TCCR1A = 0;
  TCCR1B = 0;

  TCNT1 = 3035;          // hodnota,do ktere se cita
  TCCR1B |= (1 << CS12); // precsalarni hodnota
  TIMSK1 |= (1 << TOIE1); // povoleni preruseni pri preteceni
  SREG = B10000000; //globalni povoleni preruseni
}
```

```
void loop() {

}
ISR(INT0_vect) //obsluzna rutina
{
  SREG = 0x0;
  PORTB = (stav << 0);
  stav = !stav;
  delay(100);
  SREG = B1000000;
}

ISR(TIMER1_OVF_vect)     // obsluzna rutina preruseni
{
  SREG = 0x0;
  TCNT1 = 3035;          // opetovne nastaveni hodnoty do ktere se cita
  PORTD = (pin2 << 2);   //zmena stavu pinu s led diodou
  pin2 = !pin2;
  SREG = B1000000;
}
```

Výpis 9: Externí přerušení-přímý zápis do registrů

4.1 Shrnutí

Arduino IDE je prostředí vyvinuté přímo pro mikrokontroléry Arduino. Programovacím jazykem je C/C++,čímž se řízení Arduina v tomto prostředí stává nesrovnatelněji rychlejším ve srovnání s Matlabem a Simulinkem. Nevýhodou Adruina IDE je nedostatek vhodných vizualizačních nástrojů(pouze Serial Ploter a Serial Monitor). Výhodami Arduina IDE jsou:

- Dostupnost velkého množství knihoven určených k ovládání shieldů a senzorů.
- Velka podpora komunity Arduina
- Nástroje pro vypálení bootloaderu

5 Scilab

Open source software dostupný pod CECILL licencí, určený pro numerické výpočty velmi podobný Matlabu. Je dostupný pro operační systémy Windows, Linux a Mac OS X. Scilab používá vyšší stejnojmenný programovací jazyk. K řízení arduina je na oficiálních stránkách Scilabu dostupný balíček Arduino. Po stažení balíčku stačí do Konzole Scilabu zadat příkaz **atomsInstall("arduino")** a poté příkaz **atomsLoad("arduino")**. Součástí balíčku je i soubor *toolbox_ardduino_v*3*.ino*, který se musí přes Arduino IDE, nebo jiný sériový terminál nahrát do Arduina. Komunikace Arduina a Scilabu pak probíhá přes sériovou komunikaci rychostí 115200Bd. Stejně jako v Matlabu, se do nápovědy v Scilabu přidala položka Arduino obsahující popis jednotlvých bloků pro ovládání Arduina. Prostředí Scilab v sobě integruje prostředí Xcos, které se podobá Simulinku v Matlabu, a slouží k modelování a simulaci. Xcos bude po instalaci podobně jako Simulink obsahovat bloky určené k řízení Arduina. Do prostředí Xcos se dá dostat pomocí příkazu xcos, který zadáme do konzole Scilabu, nebo pomocí ikony, nacházející se vedle ikony nastavení na panelu nástrojů. Balíček Arduino v prostředí xcoss v Palette browseru obsahuje čtyři položky[6]:

- **Configuration:** Zde nalezneme bloky:

  ARDUINO_SETUP: Tento blok je nutné vždy přidat do projektu, používáme-li Arduino. V bloku nastavujeme typ desky a port na kterém je Arduino připojeno.

  TIME_SMPLPE: V tomto bloku definujeme dobu vzorkování.

  ARDUINO_SCOPE: Blok reprezentující graf pro zobrazení průběhu hodnot.

- **Digital:** Tato položka obsahuje bloky pro čtení a zápis logické úrovně na digitální pin(blok **DIGITAL_WRITE** a **DIGITAL_READ**), dále blok **INTERRUPT_SB**, který slouží pro zaznamenávání externího přerušení na určeném pinu, a nakonec blok **ENCODER_SB**, který inkrementuje nebo dekrementuje (záleží na pořadí zaznamenaných přerušení)výstup pokaždé, když dojde k zaznamenání požadavku přerušení na určených pinech (používá se např. pro určení otáček motoru).

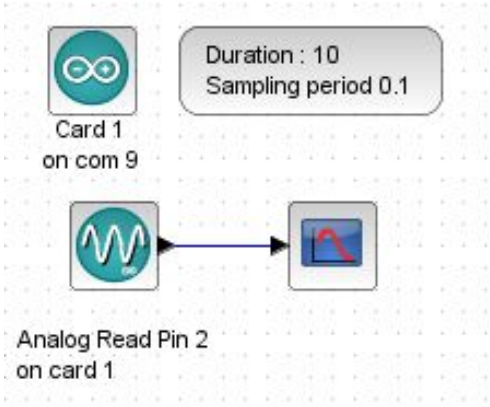
- **Analog:** Zde se nachází bloky **ANALOG_READ** a **ANALOG_WRITE**, určené pro čtení a zápis analogové hodnoty.

- **Motors:** Jsou zde bloky **DCMOTOR_SB** a **SERVO_WRITE**.

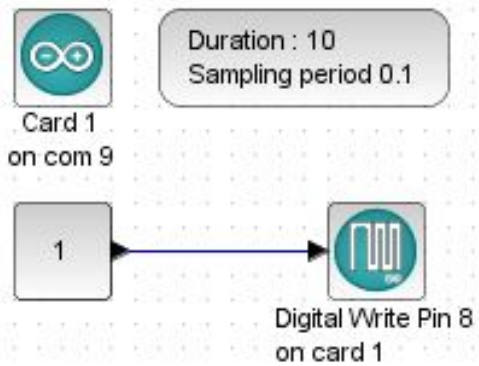
Toolbox pro Arduno neposkytuje nástroje pro plohodnotné ovládání Arduina, spíš se jedná o zjednodušení získávání dat z Arduina, ovládání digitálních pinů a PWM, což se před uvolněním tohoto toolboxu řešilo pomocí řídících příkazů posílaných přes sériovou komunikaci mezi Arduino IDE a Scilabem(vyžaduje The Scilab Serial Communication Toolbox), kdy bylo nutné definovat jednotlivé řídící příkazy. Soubor *toolbox_ardduino_v*3*.ino* definuje jednotlivé řídící příkazy v Arduino IDE a zahajuje seriovou komunikaci.

5.1 Příklady Schémat v Scilabu

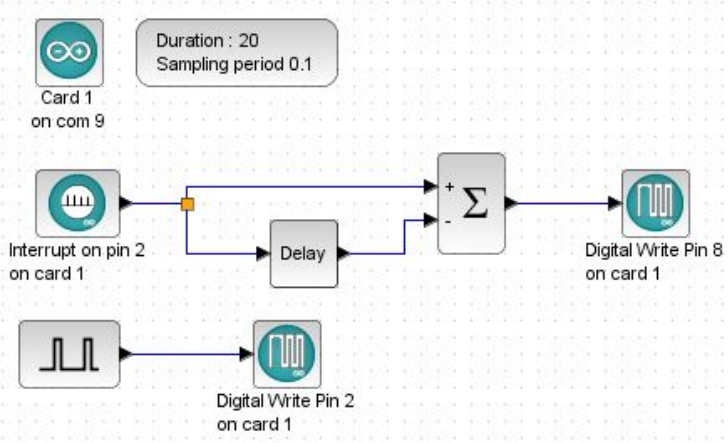
Základním prvkem schématu pro Arduino je blok **ARDUINO_SETUP**, ve kterém definujeme čislo portu, to lze najít pomocí programu Arduino IDE nebo pomocí správce zařízení ve Windows, další položkou tohoto bloku je typ desky(pro Arduinu Uno je to číslo 1). Dalším nutným prvkem je blok **TIME_SAMPLPE**, zde udáváme dobu po kterou simulace bude běžet a periodu vzorkování. Po zhotovení schématu se simulace spouští v záložce **Simulation** -> **Start**. Během Simulace se hodnoty je-li v schématu obsažen blok ARDUINO_ SCOPE zobrazují do grafu.



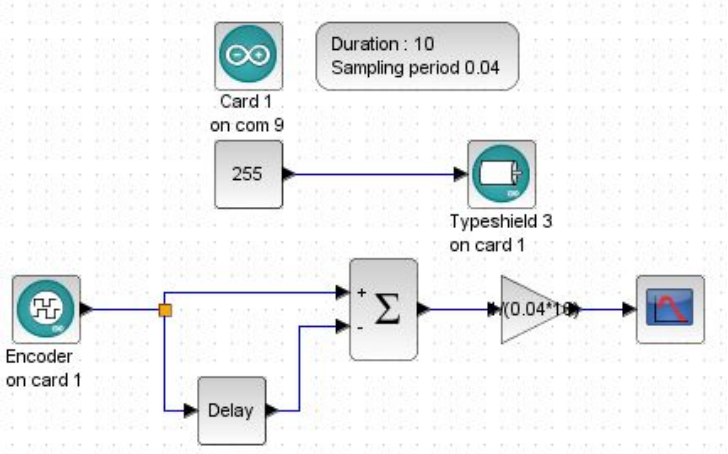
Obrázek 5: Čtení analogové hodnoty



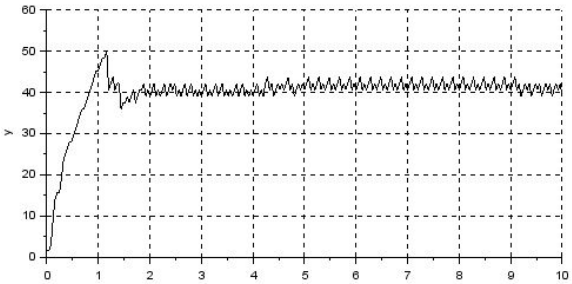
Obrázek 6: Zapsáni logicke 1 na pin 8



Obrázek 7: Ovládáni LED diody softwarovým přerušením



Obrázek 8: Schéma pro měření otáček



Obrázek 9: Ukázka vykreslení Průběhu otáček v Arduino Scope

5.2 Shrnutí

Scilab je nástroj určený pro matematické výpočty, simulace a modelování. Toolbox pro Arduino je určen pouze pro některé desky a jen pro prostředí Xcos. Toolbox obsahuje pouze základní nástroje(nástroje pro čtení a zápis analogové hodnoty, encodér, PWM) a tedy neumožňuje I2C komunikaci, 1-Wire komunikaci, připojení LCD displeje atd, což je veliký nedostatek, jelikož velká část senzorů, určená pro Arduino, komunikuje právě pomocí I2C a 1-Wire komunikace. Nevýhodou je také nutnost nahrání souboru *toolbox_ardduino_v3.ino*" přes Arduino IDE. Výhodami Scilabu použitého k implementaci kódu pro Arduino jsou:

- Možnost využití nástojů pro složité matematické operace a výpočty.
- Scilab je zcela zdarma.
- Scilab je vhodný pro zpracování a analýzu dat získaných z Arduina, kdy lze využít pokročilé nástoje Scilabu.
- Možnost vytváření schémat v prostředí Xcos namísto implementace kódu.

6 Matlab

Matlab je interaktivní programovací prostředí a také skriptovací programovací jazyk čtvrté generace pro operační systémy Windows, Linux a MacOS. Programovací jazyk Matlab vychází z programovacího jazyka Fortran. Matlab původně vytvořil profesor Cleve Moler jako nadstavbu pro usnadnění práce s knihovnami LINPACK a EISPACK pro práci s maticemi. Klíčovou datovou strukturou Matlabu jsou tedy matice, ty nevyžadují nastavení dimenzí jako v případě jiných programovacích jazyků. Zvláštností také je slabá dynamická typová kontrola, kdy proměnné nevyžadují deklaraci, vzniknou prvním přiřazením hodnoty (datový typ proměnné je tedy určen přiřazenou hodnotou). Během existence proměnné, může docházet k změně jejího datového typu v závislosti na přiřazovaných hodnotách proměnné. Změna datového typu a velikost proměnné probíhá automaticky.[8]

Matlab je tedy systém vhodný pro vědecké a inženýrské výpočty, modelování, simulace, analýzu a vizualizaci dat. Základní možnosti Matlabu rozšiřují knihovny funkcí tzv. toolboxy, soubory M-funkcí zaměřené na specifické účely (statistika, optimalizace, symbolické výpočty, neuronové sítě, zpracování signálů a obrazu, apod.). Autor MATLABu, firma The MathWorks Inc., dostupná z `http://www.mathworks.com/`, stejně jako její zástupce pro Českou republiku a Slovensko, firma HUMUSOFT,dostupná z `http://www.humusoft.cz/`, poskytuje svému prostředí značnou podporu.[8]

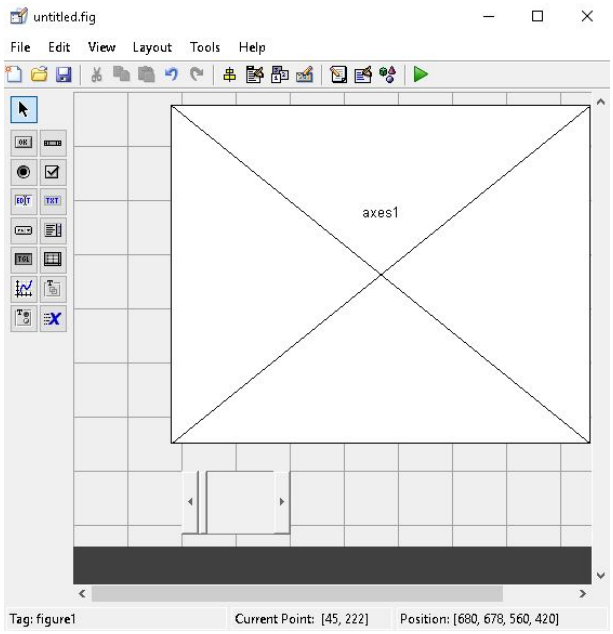
6.1 Komponenty Matlabu

- Toolboxy: Adresáře s mex a m soubory, tvořící ucelené balíky, které obsahují určitý obor, ve kterém Matlab nachází uplatnění např. Aerospace Toolbox, Fuzzy Logic Toolbox, Neural NetworkToolbox,. . .

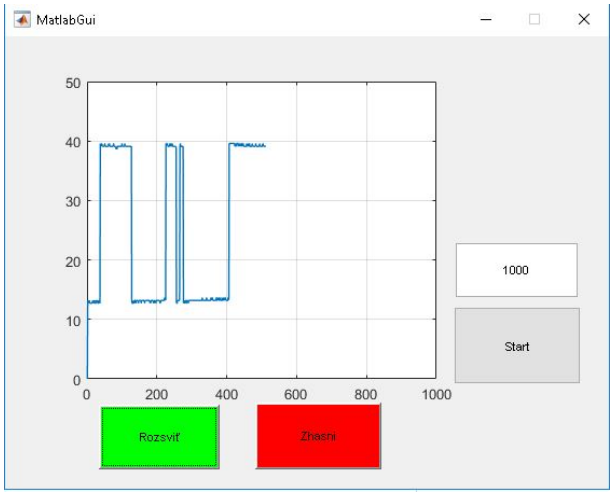
- Simulink: Nadstavba Matlabu, využívající blokových schémat. Umožnuje simulovat smíšené systémy, obsahující spojité části, diskrétní části s různými periodami vzorkování a s posunutými okamžiky vzorkování. Simulink byl původně jen knihovna určená k simulaci lineárních spojitých a diskrétních systémů, od verse Matlabu 6.5 (Matlab 2002) je Simulink samostatný subsystém s uživatelským rozhraním, které využívá blokových schémat. Základem jsou základní bloky, které nelze rozdělit na jiné bloky a které obsahují zabudované funkce tzv. S funkce, mex a m soubory. Bloky lze vytvářet pomocí skupiny bloků, uzavřených do subsystémů nebo vložením kódu v jazyce Matlab do k tomu určeného bloku (např. blok S-Function Builder). Po naistalování hardwarového balíčku pro Arduino se do Simulink Library browseru přidá položka Simulink Support Package for Arduino Hardware, obsahující bloky pro Arduino.

- Matlab GUI(graphical user interface): Poskytuje rozhraní mezi uživatelem a aplikaci podřízeným kódem (řízení kódu pomocí grafických ikon jako je slider, tlačítko,..). Toto prostředí lze spusti příkazem *guide* v Command Window Matlabu. Po vytvoření nového GUI se v

Matlabu zobrazí okno pro grafickou editaci(v tomto okně se vytváří grafická podoba aplikace pomocí ikon tlačítka, slidru, textového okna, grafu atd.) a soubor *nazev_gui.m*, ve kterém se definuje funkčnost jednotlivých grafických komponent. Vzheld okna pro grafickou editaci je na obrázku **??**. Příklad použití nástroje Matlab GUI je ukázán na obrázku **??**. V tomto příkladu je čtena a zobrazována analogová hodnota na fotorezistoru, jejíž velikost můžeme měnit spínáním LED diody, k čemuž slouží dvojice tlačítek.



Obrázek 10: Okno pro vytváření GUI



Obrázek 11: Příklad použití Matlab GUI pro vizualizaci dat z Arduina

6.2 Matlab arduino

Matlab umožňuje přímo pracovat s platformou Arduino pomocí hardwarového balíčku, který je nejprve nutné do Matlabu přidat. Tento balíček podporuje: Aruino UNo, Due, Mega 2560, Leonardo, Mega ADK. Přidání a instalaci balíčku provedeme následovně: v Matlabu otevřeme v Ovládacím panelu záložku Home, poté zvolíme rolovací menu u položky Add-Ons a vybereme položku Get Hardware Support Packages. V nově otevřené okně se zvolí možnost Install from Internet. Následně najdeme požadovaný balíček a nainstalujeme jeho části (pro arduino jsou zde dvě položky jedna pro Matlab a druhá pro Simulink).

6.2.1 Simulink Arduinu

Pro Simulink jsou dostupné dva balíčky: Arduino IO Library a Simulink Support Package for Arduino Hardware. Balíček Arduino IO Library na rozdíl od Simulink Support Package for Arduino Hardware umožnuje real-time a obsahuje navíc bloky: **encoder**, pomocí kterého je možné zaznamenávat externí přerušení a bloky pro řízení krokového motoru. Nadruhou stranu neobsahuje bloky pro sériovou komunikaci, I2C komunikaci a bloky pro Ethernet a Wifi Shliedy. Bloky jednotlivých balíčků se vkládají přes Simulink Library browser. Ukázky použití bloků obou balíčku jsou uvedeny níže. Před spuštěním vytvořeného schématu se musí nejprve nastavit v **Tools->Run on target hardware->prepare to run** v položce Target hardware typ používané desky(např. Arduino Uno). Pro nahrání kódu do Arduina slouží ikona Deploy to hardware. Na stejné liště lze také nastavit doba simulace(pro nepřetržitou simulaci je možno nastavit inf) a mód simulace. Defaultně je nastaven mód Normal. V tomto módu se čeká na dokončení simulace a teprve poté se zobrazí výsledek(v případě grafu se graf vykreslí až po dokončení simulace). Pro Simulink Support Package for Arduino Hardware lze využít také mód External, který oproti módu Normal umožňuje zobrazování hodnot během simulace. Balíček Arduino I/O je nutné spouštět pouze v módu Normal s bloky Real-Time Parec a Arduino IO Setup. V Simulinku je možnost vytvoření nového bloku pomocí S-Function Buildru, ten lze najít v Simulink Lirary Browseru v položce User-Defined Function. Pomocí tohoto bloku si lze např. vytvořit blok pro výpis znaků na LCD displej.

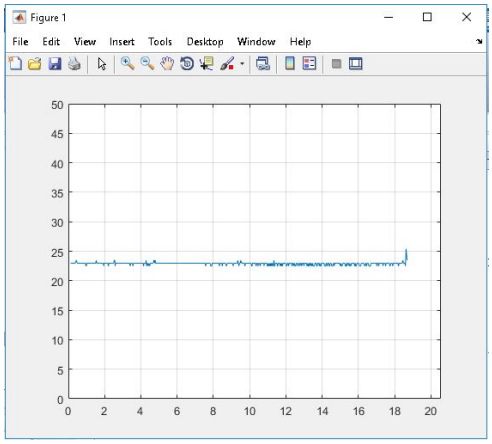
6.2.2 Základní funkce

Pokud instalace balíčku proběhla správně a balíček byl naistalován, přidala se také nápověda pro Arduino. V Helpu se na konci seznamu přidá nápověda pro práci s arduinem v Maltabu a v Simulinku. Ta obsahuje popis funkcí a několik příkladů (např. čtení analogové pinu, zápis na digitální výstup, ovládání serva). Také je zde popsán postup vytvoření vlastní knihovny pro Arduino, nicméně to vyžaduje pokročilé znalosti Matlabu.

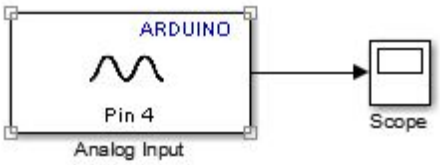
- Čtení a zápis analogové veličiny v Matlabu: V tomto příkladu je použit analogový senzor teploty LM35, který je připojen na pin A5.

```
clear all
a = arduino('COM9');%vytvori objekt arduina
tic %zde se zacina merit cas
i=0;
while(1)
i=i+1;
teplota(i)=100*readVoltage(a,'A5');%funkce cte analogovou hodnotu napeti
%na pinu A5 a prevadi na teplotu
%podle vztahu uvedeneho v datasheetu t=V/0.01

t(i)=toc;%prirazuje aktualni cas od zacatku mereni
plot(t,teplota);
%nastaveni os, casova osa je nastavena tak, aby se postupne zvetsovala svuj
%rozsah hodnot
axis([0 (t(i) + 0.1*t(i)) 0 50])
grid
drawnow;
end
```

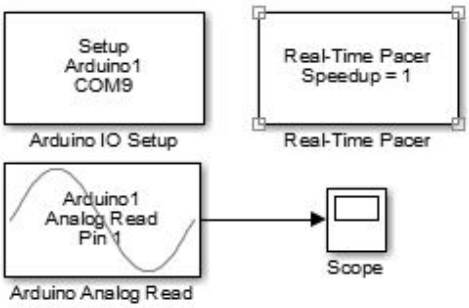
Výpis 10: Čtení analogové hodnoty a její zobrazení



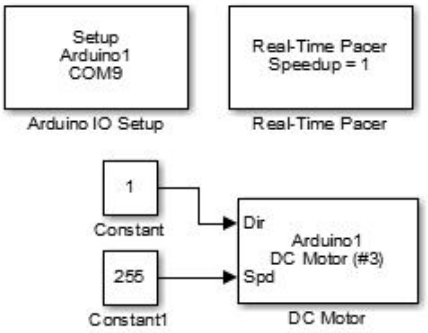
Obrázek 12: Průběh hodnot



Obrázek 13: Čtení analogové hodnoty pomocí Hardwerového balíčku pro Adruino



Obrázek 14: Čtení analogové hodnoty pomocí balíčku Arduino IO Library



Obrázek 15: Ovládání DC motoru pomocí Arduion IO Library

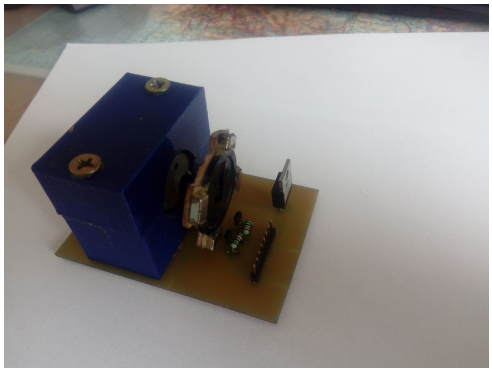
6.3 Shrnutí

Matlab je velmi robustní nástroj, nacházející uplatnění v mnoha technických oborech. Podpora v podobě softwarových balíčků pro Arduino je rozsáhlejší ve srovnání se Scilabem a nabízí možnosti rozšiření v podobě S-funkcí v Simulinku a knihoven v Matlabu. Velkou nevýhodou použití Matlabu k implementaci kódu pro platformu Arduina je jeho vysoká cena. MathWorks sice nabízí studentskou verzi, nicméně i ta je zpoplatněna (cena se pohybuje kolem $ 100). Výhodami Matlabu jsou:

- Možnost vytváření schémat V Simulinku namísto psaní kódu
- Možné využití Matab GUI pro vizualizaci
- Robustní nástroje pro zpracování a analýzu dat, matematické operace a modelovaní systémů.
- Velká podpora v podobě fóra a MathWorku.
- Na rozdíl od Arduino toolboxu ve Scilabu jsou v hardwarovém balíčku pro Matlab funkce pro I2C komunikaci, Ethernet a Wifi.

7 Řízení DC motoru

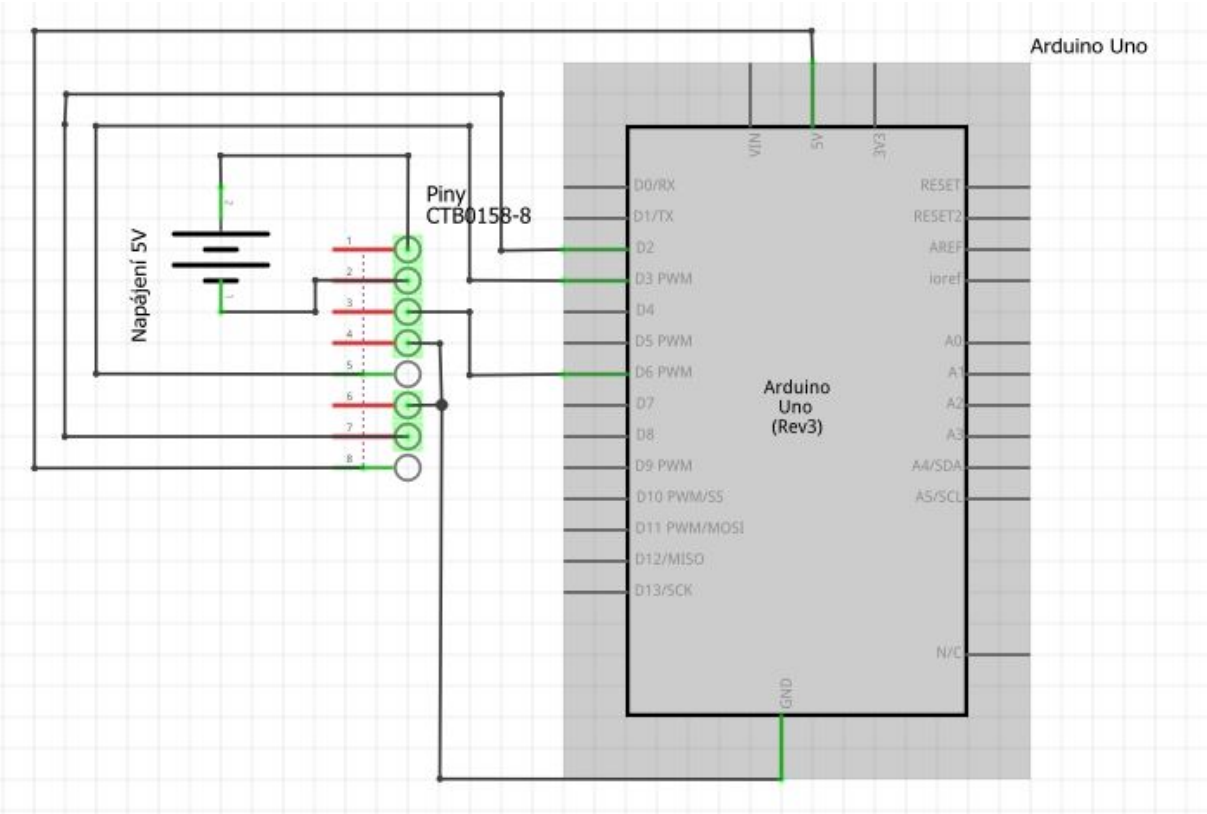
Pro realizaci regulace a identifikaci stejnosměrného motoru byl zhotoven jednoduchý model se stejnosměrným motorem viz. obrázek 16. Otáčky u tohoto modelu jsou snímány dvojicí hallových sond A1104. Otáčky jsou regulovány pomocí PWM přes MOSFET tranzistor. Napájení motorku je 5V a je nutné zajistit externí napájení.



Obrázek 16: Model se stejnosměrným motorkem

1	Externí napájení 5V
2	Externí napájení GND
3	Pin pro spínání tranzistoru
4	Hallová sonda1 GND
5	Hallová sonda1 OUTPUT
6	Hallová sonda2 GND
7	Hallová sonda2 OUTPUT
8	Halové sondy 5V

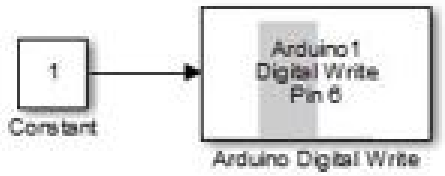
Tabulka 4: Popis pinů na modelu stejnosměrného motorku



Obrázek 17: Schéma propojení s Arduinem

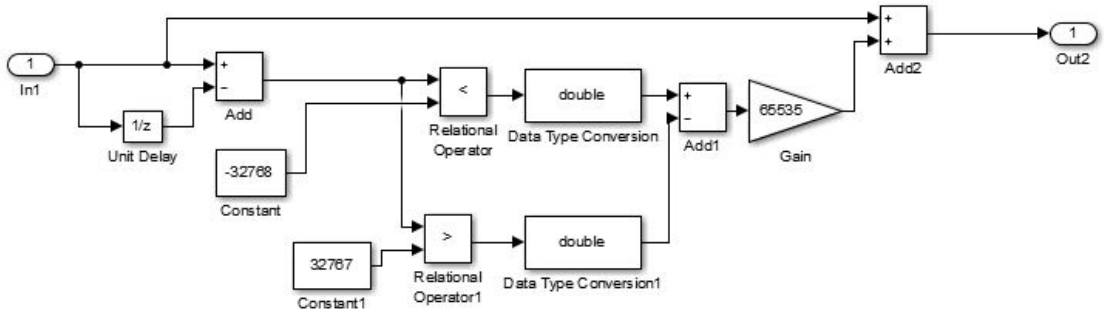
7.1 Matlab

Pro řízení motoru byla použita knihovna Arduino I/O library, umožňující řízení v reálném čase (hardwarový balíček pro Arduino neumožňuje externí přerušení a tedy nelze realizovat encodér, možnou variantou řešení je posílání dat o otáčkách z prostředí Arduino IDE). Prvním krokem v úloze řízení motoru je změření přechodové charakteristiky motoru. Skoková změna bude v našem případě připojení motoru k 5V pomocí bloku Arduino Digital Write. Pro snímání otáček



Obrázek 18: Realizace jednotkového skoku

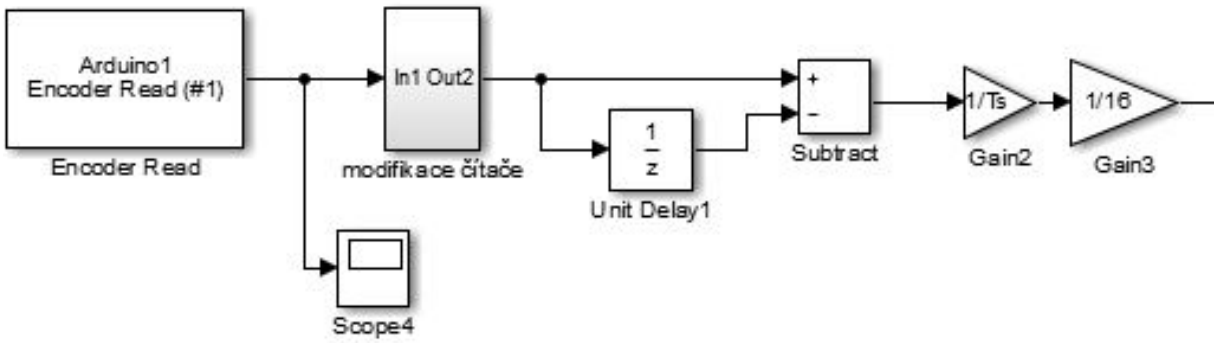
byl využit blok Encoder, který čítá pokaždé, když dojde k změně hodnoty na pinech umožňující externí přerušení (pro Uno je to pin 2 a pin 3). Tento Encodér se inkrementuje či dekrementuje v závislosti na směru otáčení, nicméně čítá pouze do hodnot 32767 a -32768, poté začíná čítat od začátku, proto je nutné výstup Encoderu upravit. Toto schéma zaručí, že hodnota udávající



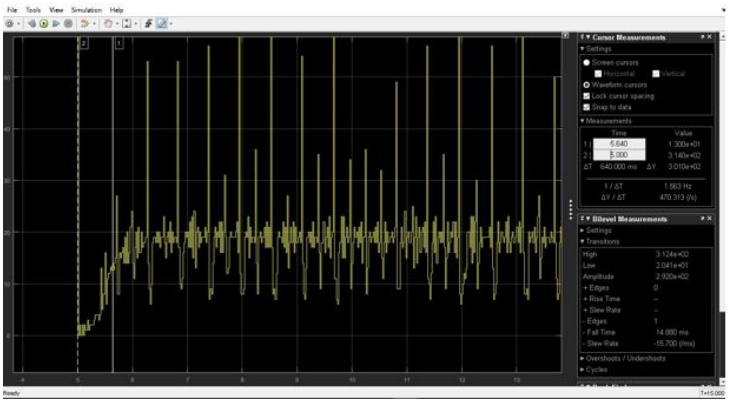
Obrázek 19: Modifikace výstupu enkodéru

počet přerušení se bude inkrementovat i po překonání limitních hodnot Encoderu. Nyní je nutné určit rychlost otáčení. Ta se určí jako diference mezi jednotlivými vzorky vydělena periodou vzorkování a počtem přerušení, ke kterým dojde v průběhu jedné otočky (v našem případě dojde k osmi přerušením během jedné otočky->dochází k detekci sestupné i vzestupné hraně na dvou hallových sondách). Výstupem bude velmi zašumělý signál. Abychom z tohoto získaného průběhu mohli určit přenos soustavy je potřeba filtrovat šum. Z průběhu přibližně odhadneme časovou konstantu, tak že odečteme čas, kdy dojde k dosažení 63,2% ustálené hodnoty. Tuto konstantu použijeme jako časovou konstantu filtru. Filtr pak bude mít přenos ve tvaru[5]:

$$G(s) = \frac{1}{Ts + 1} \tag{2}$$



Obrázek 20: Převedení modifikovaného výstupu enkodéru na otáčky

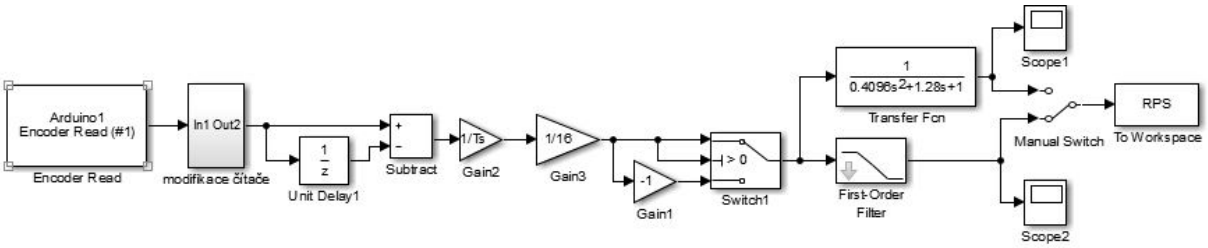


Obrázek 21: Průběh otáček

Pro náš případ byla odhadhadnuta časová konstanta na 0,64 s a tedy přenos bude:

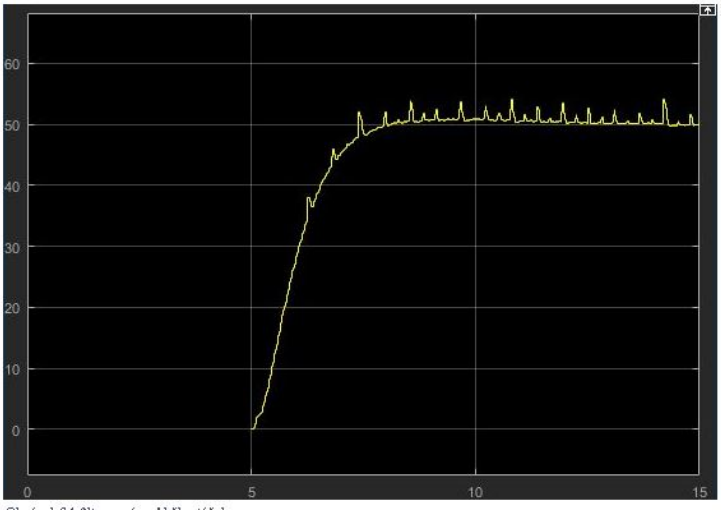
$$G(s) = \frac{1}{0,64s + 1} \tag{3}$$

Je možné použít filtr druhého řádu, nicméně je nutné vzít v úvahu, že použitím filtru dochází k zvětšování časového zpoždění. Toto dopravní zpoždění s rostoucím řádem filtru roste. V našem případě postačí filtr dolní propust 1.řádu. Pro náš filtr byl použit blok First-OrderFiltr, je možné použít blok Transfer Function, který má parametr přenos filtru. Filtrovaný výstup pak



Obrázek 22: Průběh otáček

posíláme do Workspacu pro identifikaci signálu. Dalším krokem je identifikace přenosu a následné



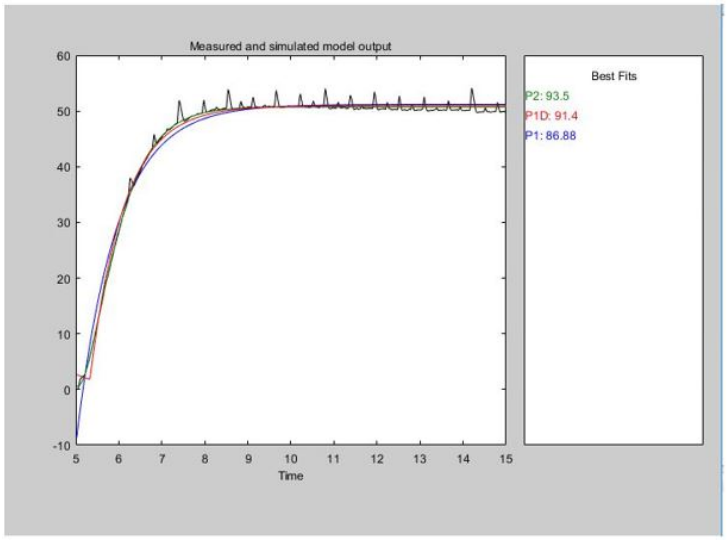
Obrázek 23: Filtrovaný průběh

stanovení konstant regulátoru. Do workspacu se nám po proběhnutí simulace přidala proměnná RPS, ta uchovává data přechodové charakteristiky motoru. Pro identifikaci přenosu použijeme nástroj ident, nejprve ale musíme vytvořit vstupní signál(konstantní signál o velikosti 5 a stejné délce jako je délka změřené přechodové charakteristiky).

```
y=RPS.Data
t=RPS.Time
for i =1:length(y)
if(y(i) >0)
    if(i ~=1)
    ynew=y(i-1:(length(y)),1)
    tnew=t(i-1:(length(t)),1)
    break
    end
    if(i ==1)
        ynew=y
        tnew=t
       break
    end
end
end
u(1:(length(ynew)),1)= 5
ident
```

Výpis 11: Úprava dat a identifikace

Po importování vstupního a výstupního signálu, nástroj ident odhadl následující přenosy.



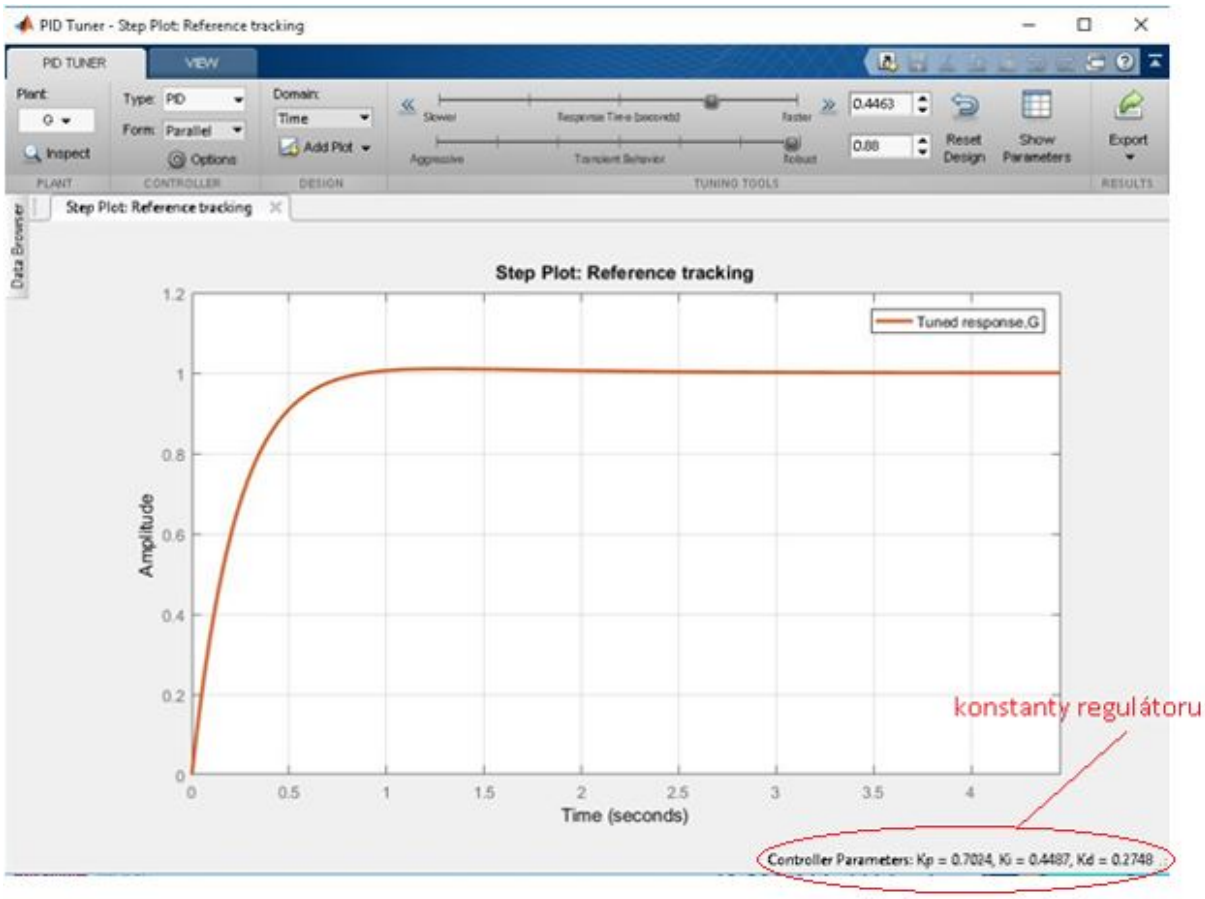
Obrázek 24: Přechodové charakteristiky odhadnutých průběhů

Z obrázku 24 je patrné, že naměřenou přechodovou charakteristiku nejlépe kopíruje přenos druhého řádu (zelený průběh).

```
Process model with transfer function:
                 Kp
  G(s) = -----------------
         (1+Tp1*s)(1+Tp2*s)

          Kp = 10.149
         Tp1 = 0.50373
         Tp2 = 0.51603
```

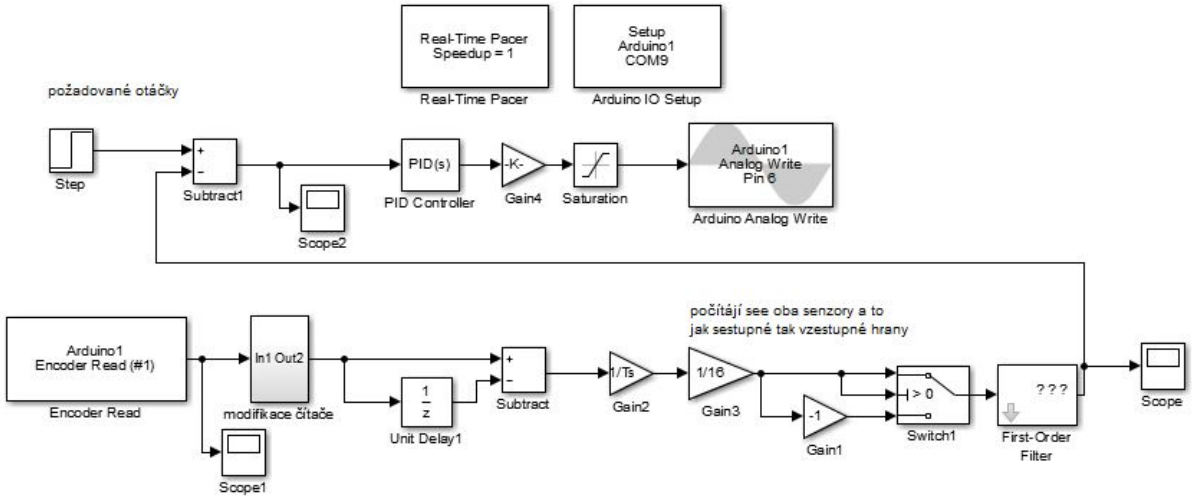
Obrázek 25: Přenos soustavy

Nyní, když máme přenos motoru, můžeme určit konstanty regulátoru. K tomu lze využít nástroj pidTuner, kde je možné navrhnout konstanty regulátoru. Pro jeho spuštění stačí zadat příkaz pidTuner, nebo příkaz pidTuner(G), kde G znamená přenos soustavy. V případě použití pidTuner(G) Matlab sám navrhne konstanty regulátoru a vypíše je do Command window.

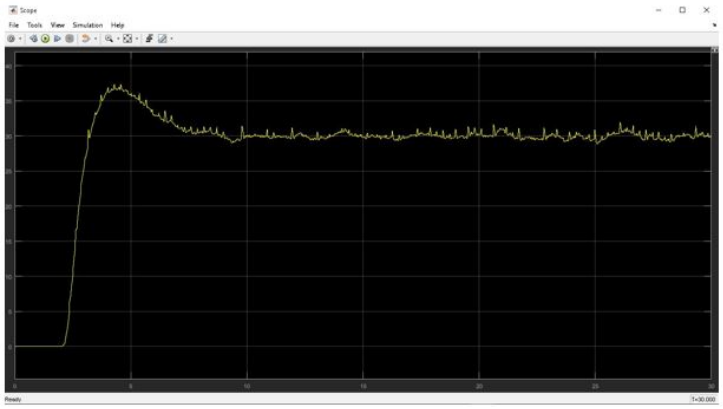


Obrázek 26: PID tuner

Posledním Krokem je vytvoření regulačního schémata. V bloku Step nastavujeme požadovanou hodnotu otáček, ve shématu na obrázku 27 je požadovaná hodnota otáček nastavena na 30 otáček za sekundu.



Obrázek 27: Schéma regulačního obvodu

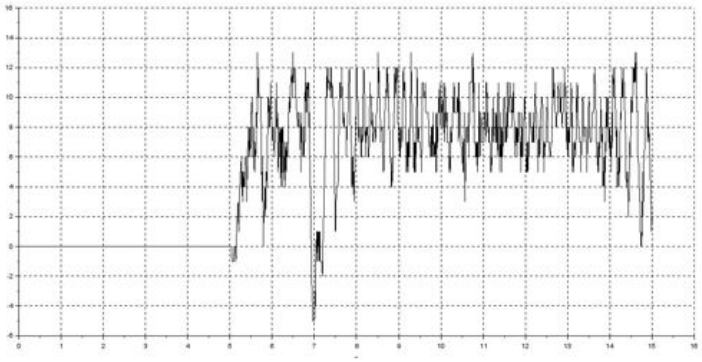


Obrázek 28: Regulovaný průběh otáček

7.2 Scilab

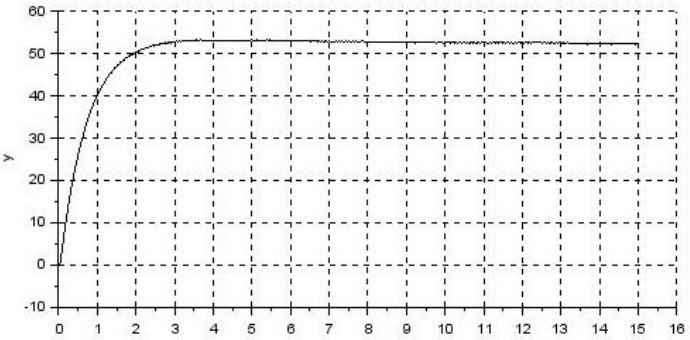
V Scilabu byl použit balíček Arduino. Pro používání tohoto balíčku je nezbytné nahrání souboru toolbox_arduino_v3 do arduina přes prostředí Arduino IDE. Ke snímání otáček poslouží blok Encoder, který reaguje na přerušení na definovaném pinu (pro Arduino Uno je to pin 2 a 3) nebo blok Intterupt on pin 2. Oba tyto bloky inkrementují výstup pokaždé, když dojde k detekci externího přerušení na daném pinu. K získání rychlosti lze využít bloku Delay a sumačního bloku, kdy podobně jako v prostředí Matlab&Simulink je rychlost určena jako diferenci mezi jednotlivými vzorky. Pro získání rychlosti otáčení je nutné průběh diferencí vynásobit převrá-

cenou hodnotou periody vzorkování a vydělit počtem vygenerovaných přerušení během jedné otočky. Průběh otáček je zašumělý, a proto je potřeba tento průběh nejprve filtrovat. K filtraci

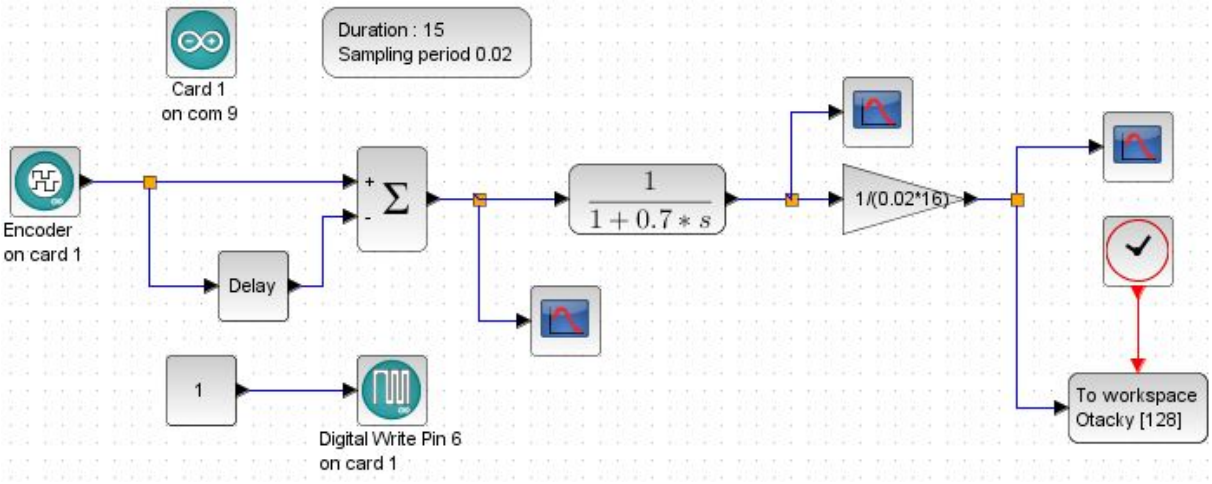


Obrázek 29: Průběh diferencí mezi jednotlivými vzorky

postačí dolnopropustný filtr 1.řádu (filtry vyšších řádu mnohem více zvětšují dopravní zpoždění signálu). Časovou konstantu určíme odhadem, jako čas, kdy dojde k dosažení 63,3% ustálené hodnoty průběhu.



Obrázek 30: Filtrovaná přechodová charakteristika



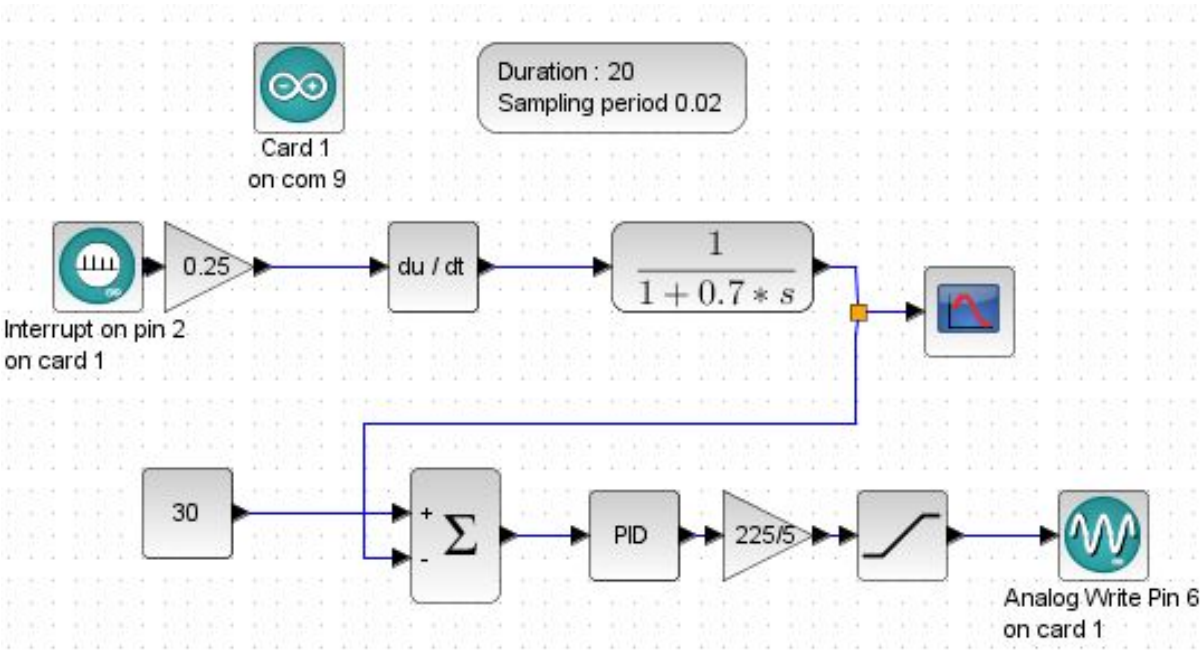
Obrázek 31: Schéma pro zmeření přechodové charakteristiky

Z Filtrovaných dat o průběhu rychlosti otáčení určíme přenos motoru pomocí metody ploch ,pomocí které vypočteme koeficienty přenosu. Zesílení se pak vypočte jako poměr mezi ustálenou hodnotou a hodnotou vstupního skokového signálu. K určení konstant PID regulátoru byla zvolena Kuhnova metoda, také známá jako metoda souhrnné časové konstanty.

```
i=1;
//nalezeni zacatku prechodove charakteristiky
while(Otacky.values(i) == 0)
    start_time=i;
    i = i +1;
end

for i=start_time:length(Otacky.values)
    y(i-start_time+1)=Otacky.values(i);
    t(i-start_time+1)=Otacky.time(i);
    u(i-start_time+1)=5;
end
//identifikace

//metoda ploch
radSystemu=2;
y_norm=y./y(length(y));//normovana data
for j=1:radSystemu
    Momenty(j)=inttrap(t,(1-y_norm).*(-t).^(j-1)/factorial(j-1));
end


for  i =1:radSystemu
if i == 1
     S(1)=Momenty(1);
```

```
elseif i >=2
    for  k=1:(i-1)
        M=i-k
    S(i)=S(k)*Momenty(M);
end
end
S(i)=S(i)+Momenty(i);
a(i)=S(i)/5;
end

Zesileni=y(length(y))/5;
//navrh regulatoru metoda kuhnova(souhrna konstanta)
A=inttrap(t,(1-y_norm));
Tc=A
Kp=1/Zesileni
Ki=0.66*Tc
Kd=0.167*Tc
```

Výpis 12: Identifikace přenosu a navržení PID konstant



Obrázek 32: Schéma pro regulaci otáček

7.3 Arduino IDE

Program je koncipovámn podobně jako stavový automat. Tedy je funkčně rozdělen na stavy. Tyto stavy jsou definované jako výčtový typ a přepínání mezi jednotlivými stavy je realizováno pomocí funkce Switchs, která je opakovaně volána ve funkci loop. Časování je realizováno pomocí časovače Timer1 a Timer2. Pro jejichž implementaci byly použity knihovny TimerOne a MSTimer2. Otáčky se snímají pomocí externího přerušení.

Stavy:

- **nastavení:** zde dojde k nastavení čaosvačů a externího přerušení pro pin 2, na který je připojený hallův senzor. Po naměření přechodové charakteristiky dojde k zastvení časovčů a zablokování externího přerušení. Následně je nastaven stav ident.
- **ident:** V tomto stavu jsou volány funkce pro identifikaci přenosu systému a nastavení PID konstant. Pro identifikaci je použita metoda ploch a pro určení PID konstant byla zvolena Kuhnova metoda.Tyto metody jsou popsány dále.
- **nastaveníPID:** Vypočtené PID konstanty jsou přiřazeny do proměných regulátoru.
- **pid:** Optět se povolí externí přerušení a časovače, v tomto režimu je spuštěna metoda pro regulaci. V tomto stavu program běží, dokud nedojde k vypnutí mikrokontroléru nebo jeho přehrání.

Metoda ploch: jedná se o aproximační metodu, která aproximuje přechodovou charakteristiku ve tvaru[10]:

$$G(s) = \frac{k}{a_n s^n + a_{n-1} s^{n-1} + ... + a_1 s + 1} \tag{4}$$

Kde K značí zesílení, jenž se spočte jako[10]:

$$K = \frac{y(\infty)}{u(\infty)} \tag{5}$$

Koeficienty polynomu ve jmenovateli přenosu se vypočítají pomocí tzv. momentů[10].

$$M_i = \int_0^\infty [K - h(t)] \frac{(-t)^i}{i!} dt \tag{6}$$

Počet momentů musí být roven řádu systému. Pomocí vypočtených momentů vypočteme plochy, v případě záporné plochy se musí výpočet ukončit. Z toho vyplývá, že maximální řád systémů je určen indexem poslední kladné plochy[10].

$$S_i = M_{i-1} + \sum_{j=0}^{i-1} S_{i-j-1} \cdot M_j \tag{7}$$

Z vypočtených ploch se koeficienty získají nasledovně[10]

$$a_i = K \cdot S_i \tag{8}$$

Jelikož pracujeme s diskrétními daty, byla pro integraci použitá lichoběžníková numerická metoda integrace. Tato metoda aproximuje integraci na konečném intervalu pomocí rozdělení polchy na lépe spočítatelné části, v našem případě lichoběžníky.

$$\int_a^b f(x)dx = \frac{b-a}{2N} \sum_{j=0} N(f(x_n) + f(X_n + 1)) \tag{9}$$

```
double* metoda_ploch(float data[], int Size, unsigned int periodavzorkovani, byte rad_systemu) {
  double M[rad_systemu], K = 0, A = 0;
  double *S;
  if ((S = (double *)calloc((rad_systemu), sizeof(double))) == NULL) {
    Serial.println("Nedostatek pameti heap");
  }
  K = Y_v_nekonecnu(data, Size);
  ////////vypocet momentu/////////
  for (int i = 0; i <= rad_systemu; i++) {
    for (int j = 0; j <= Size - 1; j++) {
      //pro integraci je pouzita trapezova metoda pro vypocet urciteho integralu
      A += ((K - data[j]) * pow(-j * (double(PERIODA) / 1000000), i) / double(faktorial(i))) +
           ((K - data[j + 1]) * pow(-j * (double(PERIODA) / 1000000), i) / double(faktorial(i)));
    }
    M[i] = A * (float(PERIODA) / 2000000);
   // Serial.println("plochy MI :" + String(M[i]));
    Serial.println("zesileni " + String(K / 5));
    A = 0;
    //////vypocet jednotlivych ploch///////
    S[0] = M[0] / K;
    if (i >= 1) {
      for (int k = i; k >= 1; k--) {
        S[i] += S[k - 1] * M[i - k];
      }
      S[i] += M[i];
      S[i] /= K;
    }
  }
  return S;
}
```

Výpis 13: Metoda ploch v Arduinu IDE

Kuhnova metoda Metoda byla odvozena v roce 1995. Někdy je známá jako metoda souhrnné časové konstanty. Konstanty PID se u této metody určují z přechodové charakteristiky soustavy a to tak, že nejprve určíme zesílení soustavy viz rovnice 5 a plochu A viz. rovnice

10. Z plochy A pak vypočteme souhrnnou časovou konstantu. Z tabulky pak určíme příslušné konstanty.

$$A = \int_0^{\infty} (K - h(t))dt \tag{10}$$

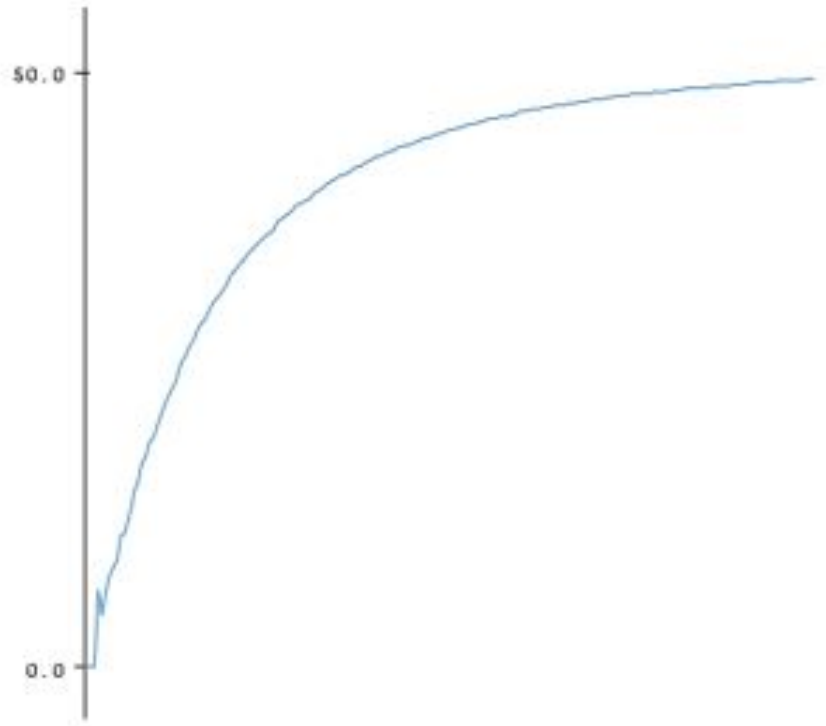
$$T_c = \frac{A}{K} \tag{11}$$

Regulátor	r_0	T_i	T_d
P	$\frac{1}{K}$		
PI	$\frac{2}{K}$	$0,7T_c$	
PID	$\frac{2}{K}$	$0,8T_c$	$0,194T_c$

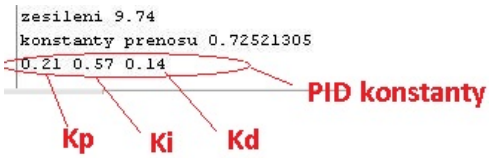
Tabulka 5: Konstanty PID-Kuhnova metoda

```
float* KuhnovaMetoda(double K, float pole[], int pocet) {
  //funkce nastavuje konstanty PID regulatoru pomoci vypoctu plochy A
  float *p, A = 0, Tc;
  if ((p = (float*)malloc(3 * sizeof(float))) == NULL) {
    Serial.println("Nedostatek pameti heap");
  }
  for (int i = 0; i < pocet - 1; i++) {
    A += (1 - (data[i]/K)) + (1 - (data[i + 1]/K));
  }
  A *= (float(PERIODA) / 2000000);
  Tc = A ;
  p[0] = 2 / (float(K)/5);
  p[1] = 0.8 * Tc;
  p[2] = 0.194 * Tc;
  //Serial.println(String(Tc) + String(K));
  return p;
}
```

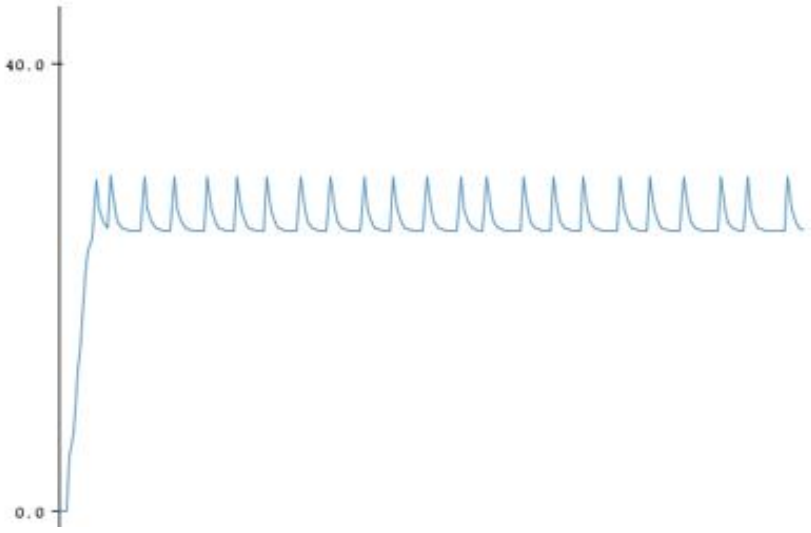
Výpis 14: Kuhnova metoda v Arduinu IDE



Obrázek 33: Přechodová charakteristika



Obrázek 34: Parmetry přenosu



Obrázek 35: Průběh regulovaných otáček

8 Závěr

Cílem této bakalářské práce bylo seznámení s mikrokontrolérem Arduinu Uno a s možnostmi jeho programování v prostředí Arduino IDE, Matlab& Simulink a Scilab. V práci byl uveden příklad identifikace a regulace soustavy stejnosměrného motoru, jenž byl realizován v jednotlivých prostředích. Pro tento příklad byl také zhotoven model se stejnosměrným motorem.

Nástoroje pro implementaci kódu pro mikrokontrolér Arduino Uno v prostředí Scilab jsou určeny pouze pro základní operace a proto je toto prostředí vhodné spíše pro zpracování a analýzu dat získaných z Arduina, než pro jeho řízení. Prostředí Matlab&Simulink je prostředí podobné Scilabu, nicméně podpora mikrokontroléru Arduina je v tomto prostředí větší, než je to v případě prostředí Scilab, a je zde také integrován nástroj Matlab GUI, který lze využít pro vizualizaci. Matlab&Simulink a Scilab disponují nástroji pro řešení složitých matemtických oprací a je tedy vhodné volit tato prostředí v případech, kdy se v řídícím algoritmu vyskytují složité matematické operace. Prostředí Arduino IDE je nejpoužívanější prostředí pro implementaci kódu pro mikrokontroléry Arduino. Toto prostředí se jeví, i přes absenci pokročilejších nástrojů pro vizualizaci, jako nejuniverzálnější. Jeho univerzálnost dle mého názoru spočívá v značném množství knihoven, určených přímo pro mikrokontroléry Arduino, které programování mikrokontroléru velmi zjednodušují a podpoře většiny desek Arduina. Další výhodou tohoto prostředí je možnost přímého zápisu do registrů, což umožňuje absolutní kontrolu a rychlejší vykonávání programu.

Zvolený příklad byl z hlediska náročnosti nejlépe realizovatelný v prostředí Matlabu, jelikož zde šlo využít nástroje PID tuner pro navržení konstant regulátoru a podobně jako v prostředí Scilab i zde byl algoritmus řízení realizován pomocí bloků, které jsou mnohem přehlednější než kód v Arduinu IDE.

Literatura

[1] SELECKÝ, M. *Arduino Uživatelká příručka* [online]. 1st ed. Brno: Computer Press, 2016 [cited 17 Jan 20]. Dostupné z : http://knihy.cpress.cz. ISBN 978-80-251-4849-5.

[2] Metody Seřizování PID Regulátorů. *www.upc.cz* [online]. [cit. 2017-04-24]. Dostupné z: http://dspace.upce.cz/bitstream/handle/10195/61154/CechZ_ MetodySerizovani_ LK_ 2015.pdf?sequence=1& isAllowed=y.

[3] Arduino Support from MATLAB. *MathWorks* [online]. [cit. 2017-04-24]. Dostupné z: https://www.mathworks.com/hardware-support/arduino-matlab.html?s_ tid=srchtitle

[4] *ATmega328 datasheed-Atmel* [online]. [cit. 2017-04-24]. Dostupné z: http://www.atmel.com/Images/Atmel-42735-8-bit-AVR-Microcontroller-ATmega328-328P_ Datasheet.pdf

[5] Simulink ArduinoIO Package. *Control Tutorials for Matlab&Simulink* [online]. [cit. 2017-04-24]. Dostupné z: http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_ IOpack.

[6] Scilab. *Scilab* [online]. [cit. 2017-04-24]. Dostupné z: http://www.scilab.org/scilab/about

[7] Ukázka části Intel-hex souboru s popisem. In: *Wikipedie* [online]. [cit. 2017-04-24]. Dostupné z: https://cs.wikipedia.org/wiki/Intel_ HEX# /media/File:Priklad_ intel_ hex.png

[8] KOLÁČEK, Jan a Kateřina KONEČNÁ. Jak pracovat s MATLABem. In: *Univerzita Pardubice* [online]. [cit. 2017-01-20]. Dostupné z: https://www.math.muni.cz/ kolacek/vyuka/-vypsyst/navod.pdf.

[9] Arduino uno. *Arduino* [online]. ARDUINO AG TRADEMARKS, http://www.arduino.org [cit. 2017-01-01]. Dostupné z: http://www.arduino.org/products/boards/arduino-uno

[10] NOSKIEVIČ, Petr. *Modelování a identifikace systémů*. Ostrava: Montanex, 1999. ISBN 8072250302.